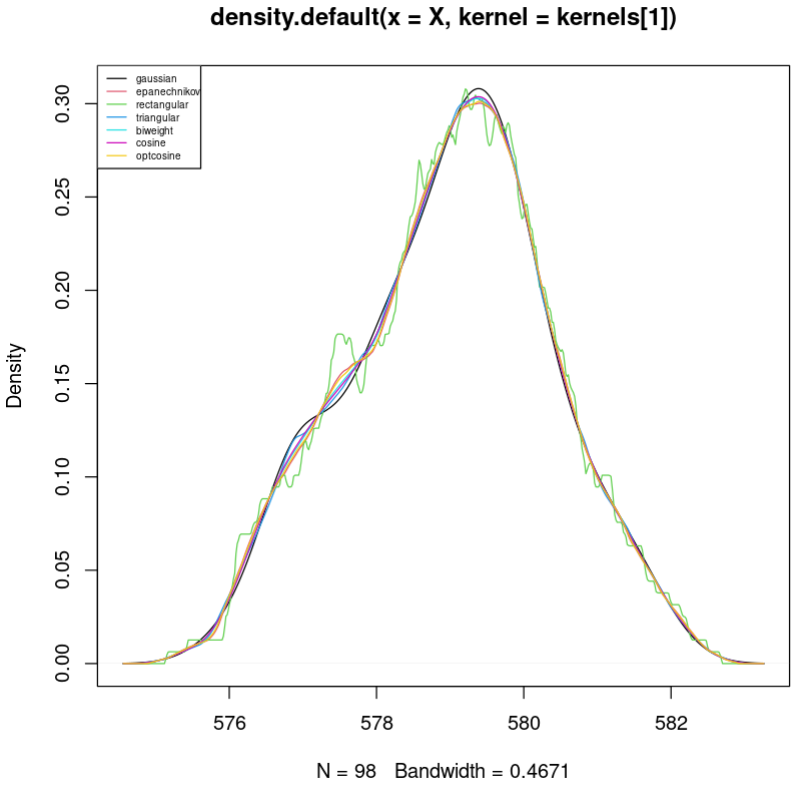
Задача N1

In [1]:
```
X = LakeHuron
```

Построим гистограммы и KDE

In [2]:
```
n_bars = c(5, 15, 25, 35, 45)
hists = vector(mode="list", length=length(n_bars))
for (k in 1:length(n_bars)) {
        hists[[k]] = hist(X, breaks=seq(from=min(X), to=max(X), length=n_
}
```

In [3]:
```
kernels=eval(formals(density.default)$kernel)
plot(density(X, kernel=kernels[1]))
for (k in 2:length(kernels)){
  lines(density(X, kernel=kernels[k]),col=k)
}
legend("topleft",legend=kernels,col=1:length(kernels),cex=0.5,lty=1)
```



In [4]:
```
kernels=eval(formals(density.default)$kernel)
kdes = vector(mode="list", length=length(kernels))
for (k in 1:length(kernels)){
  kdes[[k]] = density(X, kernel=kernels[k])
}
```

In [5]:
```
get_p_kde = function(x) {
        aux = cbind(KDE$x,KDE$y)
        aux=aux[which(aux[,1]<x),]
        return(tail(aux, n=1)[2])
```

```
}
KDE = kdes[[1]]
get_p_kde(577)
```

0.126792398926893

Посчитаем метрики

In [7]:

```
best_kde_id = 0
best_hist_id = 0
best_metric = 1000000
print(c('hist_id', 'kde_id', 'metric'))
for (hist_id in 1:length(hists)) {
        for (kde_id in 1:length(kdes)) {
                H = hists[[hist_id]]
                KDE = kdes[[kde_id]]
                centers = H$mids
                M = length(centers)
                p_H = H$density
                p_KDE = sapply(centers, get_p_kde)
                metric = sum((p_H - p_KDE)^2) / M
                print(c(hist_id, kde_id, metric))
                if (metric < best_metric) {
                        best_metric = metric
                        best_kde_id = kde_id
                        best_hist_id = hist_id
                }
        }
}
```

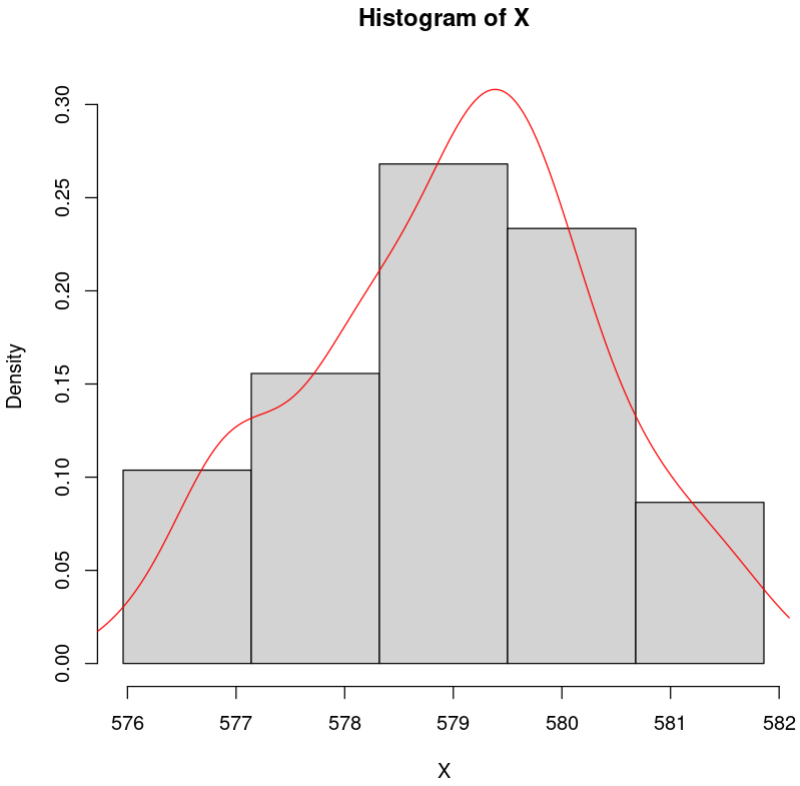
```
[1] "hist_id" "kde_id"  "metric"
[1] 1.000000e+00 1.000000e+00 6.561107e-05
[1] 1.000000e+00 2.000000e+00 7.337328e-05
[1] 1.000000e+00 3.000000e+00 9.505023e-05
[1] 1.000000e+00 4.000000e+00 9.605794e-05
[1] 1.000000e+00 5.000000e+00 7.447549e-05
[1] 1.000000e+00 6.000000e+00 7.424845e-05
[1] 1.000000e+00 7.000000e+00 7.428999e-05
[1] 2.000000000 1.000000000 0.002548851
[1] 2.000000000 2.000000000 0.002930989
[1] 2.00000000 3.00000000 0.00335582
[1] 2.000000000 4.000000000 0.002643151
[1] 2.000000000 5.000000000 0.002807884
[1] 2.000000000 6.000000000 0.002760416
[1] 2.000000000 7.000000000 0.002885944
[1] 3.000000000 1.000000000 0.005837796
[1] 3.000000000 2.000000000 0.006220351
[1] 3.00000000 3.00000000 0.00659785
[1] 3.00000000 4.00000000 0.00585119
[1] 3.000000000 5.000000000 0.006092078
[1] 3.000000000 6.000000000 0.006051356
[1] 3.000000000 7.000000000 0.006165077
[1] 4.000000000 1.000000000 0.008041692
[1] 4.000000000 2.000000000 0.008465711
[1] 4.000000000 3.000000000 0.009235435
[1] 4.000000000 4.000000000 0.008011751
[1] 4.000000000 5.000000000 0.008321546
[1] 4.000000000 6.000000000 0.008277915
[1] 4.000000000 7.000000000 0.008408556
[1] 5.000000000 1.000000000 0.009604605
[1] 5.0000000 2.0000000 0.0100781
[1] 5.00000000 3.00000000 0.01100855
[1] 5.000000000 4.000000000 0.009644937
[1] 5.000000000 5.000000000 0.009934539
[1] 5.000000000 6.000000000 0.009880854
[1] 5.00000000 7.00000000 0.01001568
```

In [8]:
```
print(c('best_hist_id', 'best_kde_id', 'best_metric'))
print(c(best_hist_id, best_kde_id, best_metric))
```

```
[1] "best_hist_id" "best_kde_id"  "best_metric"
[1] 1.000000e+00 1.000000e+00 6.561107e-05
```

Лучшая пара:

In [10]:
```
plot(hists[[best_hist_id]], freq=F, ylim=c(0,max(kdes[[best_hist_id]]$y))
lines(kdes[[best_hist_id]], type='l', col='red')
```



Задача T1. Пункт 1

Пусть $\hat{b} = \max(X_1, \dots X_n),\ \hat{a} = \min(X_1, \dots X_n)$. Найдем $\mathbb{E}[\hat{b}], \mathbb{E}[\hat{a}]$.

$$P(\max(X_1, \dots X_n < x)) = P(X_1 < x, \dots X_n < x) = P(X_1 < x)^n = \left(\frac{x-a}{b-a}\right)^n \mathbb{I}[$$

$$p_{\max}(x) = \frac{n(x-a)^{n-1}}{(b-a)^n}\mathbb{I}[\dots]$$

$$\mathbb{E}[\hat{b}] = \int_{\mathbb{R}} x p_{\max}(x) dx = \int_a^b \frac{n(x-a)^{n-1}}{(b-a)^n} dx = \frac{a+bn}{n+1}$$

Далее воспольуемся тем, что $\mathbb{E}[\hat{a}] = b + a - \mathbb{E}[\hat{b}]$. Пользуясь этим получаем, что

$$\text{Bias}(\hat{I}_n) = \mathbb{E}[\hat{b} - \hat{a} - (b-a)] = 2(\mathbb{E}[\hat{b}] - b) = 2\frac{a-b}{n+1}$$

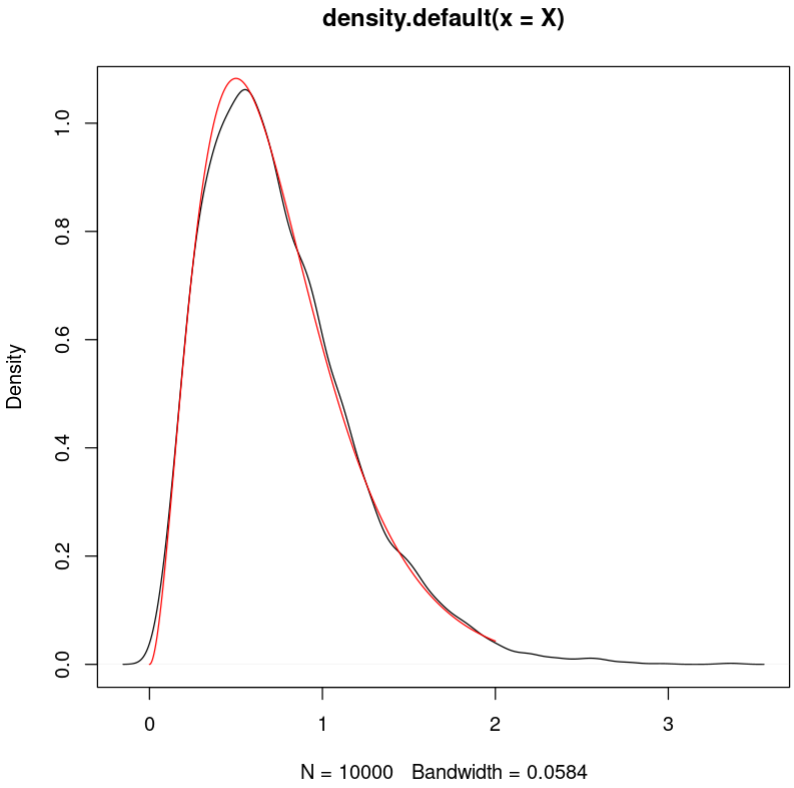
Задача N2

Пункт 1

In [14]:
```
n = 10000
X = rgamma(n, shape=3, rate=4)
```

In [12]:
```
get_p_true = function(x) {
        dgamma(x, shape=3, rate=4)
}
```

In [15]:
```
plot(density(X))
u=seq(from=0,to=2,length=1000)
lines(sapply(u, get_p_true)~u, type='l', col='red')
```



Пункт 2

In [16]:
```
get_p_kde = function(x) {
        aux = cbind(kde$x,kde$y)
        aux=aux[which(aux[,1]<x),]
        return(tail(aux, n=1)[2])
}
```

In [17]:
```
bandwidths = seq(from=0.1,to=5,by=0.1)
u = seq(from=0, to=2, by=0.01)
best_mise = 10000
for (k in 1:length(bandwidths)) {
        kde = density(X, kernel='epanechnikov', bw=bandwidths[k])
        p_pred = sapply(u, get_p_kde)
        p_true = sapply(u, get_p_true)
        mise = sum((p_pred - p_true)^2) / length(u)
        if (mise < best_mise) {
                best_mise = mise
        }
        print(c(bandwidths[k], mise))
}
```

```
[1] 0.100000000 0.001488114
[1] 0.200000000 0.009365649
[1] 0.30000000 0.02454755
[1] 0.40000000 0.04449888
[1] 0.50000000 0.06639476
[1] 0.60000000 0.08796616
[1] 0.7000000 0.1079698
[1] 0.8000000 0.1257496
[1] 0.90000 0.14126
[1] 1.0000000 0.1548384
[1] 1.1000000 0.1669819
[1] 1.2000000 0.1779267
[1] 1.3000000 0.1878691
[1] 1.4000000 0.1969278
[1] 1.5000000 0.2052171
[1] 1.6000000 0.2128162
[1] 1.7000000 0.2198087
[1] 1.8000000 0.2262519
[1] 1.9000000 0.2322074
[1] 2.000000 0.237728
[1] 2.1000000 0.2428465
[1] 2.2000000 0.2476122
[1] 2.3000000 0.2520535
[1] 2.4000000 0.2562039
[1] 2.5000000 0.2600879
[1] 2.6000000 0.2637303
[1] 2.7000000 0.2671517
[1] 2.8000000 0.2703703
[1] 2.9000000 0.2734049
[1] 3.0000000 0.2762691
[1] 3.100000 0.278976
[1] 3.2000000 0.2815391
[1] 3.3000000 0.2839689
[1] 3.4000000 0.2862753
[1] 3.5000000 0.2884672
[1] 3.6000000 0.2905526
[1] 3.7000000 0.2925392
[1] 3.8000000 0.2944338
[1] 3.9000000 0.2962421
[1] 4.0000000 0.2979701
[1] 4.100000 0.299623
[1] 4.2000000 0.3012057
[1] 4.300000 0.302722
[1] 4.4000000 0.3041766
[1] 4.5000000 0.3055729
[1] 4.6000000 0.3069131
[1] 4.7000000 0.3082026
[1] 4.800000 0.309443
[1] 4.9000000 0.3106371
[1] 5.0000000 0.3117874
```

In [18]:
```
print('best_mise')
print(best_mise)
```

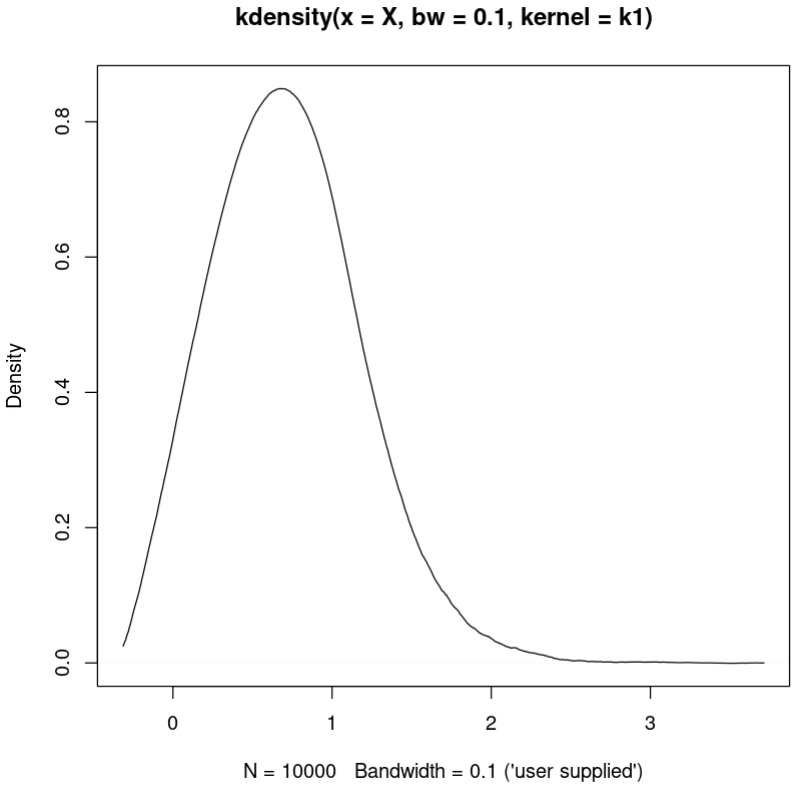
```
[1] "best_mise"
[1] 0.001488114
```

Пункт 3

In [19]:
```
install.packages("kdensity")
library(kdensity)
```

```
Installing kdensity [1.1.0] ...
        OK [linked cache]
```

In [60]:
```
inside = function(x) {
        abs(x) <= 1
}
phi0 = function(x) {
        1 / sqrt(2)
}
phi1 = function(x) {
        sqrt(3 / 2) * x
}
phi2 = function(x) {
        sqrt(5 / 8) * (3 * x^2 - 1)
}
legandre_kernel = function(y, x, h) {
        t = abs(y - x)
        (phi0(0) * phi0(t) + phi1(0) * phi1(t) + phi2(0) * phi2(t)) * ins
}
k1 <- list(
        kernel = legandre_kernel,
        support = c(-Inf, Inf)
)
out = kdensity(X, kernel = k1, bw = 0.1)
plot(out)
```



In [ ]:
```
get_p_kde = function(x) {
        aux = cbind(kde$x,kde$y)
        aux=aux[which(aux[,1]<x),]
        return(tail(aux, n=1)[2])
}
```

In [ ]:
```
bandwidths = seq(from=0.1,to=5,by=0.1)
u = seq(from=0, to=3, by=0.01)
best_mise = 10000
for (k in 1:length(bandwidths)) {
        kde = kdensity(X, kernel = k1, bw=bandwidths[k])
        p_pred = sapply(u, kde)
        p_true = sapply(u, get_p_true)
        mise = sum((p_pred - p_true)^2) / length(u)
        if (mise < best_mise) {
                best_mise = mise
        }
        print(c(bandwidths[k], mise))
}
```

```
[1] 0.1000000 0.0139428
[1] 0.2000000 0.0139428
[1] 0.3000000 0.0139428
[1] 0.4000000 0.0139428
[1] 0.5000000 0.0139428
[1] 0.6000000 0.0139428
[1] 0.7000000 0.0139428
[1] 0.8000000 0.0139428
[1] 0.9000000 0.0139428
[1] 1.0000000 0.0139428
[1] 1.1000000 0.0139428
[1] 1.20000000 0.01394356
[1] 1.30000000 0.01394356
[1] 1.40000000 0.01394356
[1] 1.50000000 0.01394356
[1] 1.60000000 0.01394356
[1] 1.70000000 0.01394356
[1] 1.80000000 0.01394356
[1] 1.90000000 0.01394355
[1] 2.00000000 0.01394355
[1] 2.10000000 0.01394355
[1] 2.20000000 0.01394355
[1] 2.3000000 0.0139437
[1] 2.4000000 0.0139437
[1] 2.5000000 0.0139437
[1] 2.6000000 0.0139437
[1] 2.7000000 0.0139437
[1] 2.8000000 0.0139437
[1] 2.9000000 0.0139437
[1] 3.0000000 0.0139437
[1] 3.1000000 0.0139437
[1] 3.2000000 0.0139437
[1] 3.3000000 0.0139437
[1] 3.4000000 0.0139437
[1] 3.5000000 0.0139437
[1] 3.6000000 0.0139437
[1] 3.7000000 0.0139437
[1] 3.80000000 0.01394367
[1] 3.90000000 0.01394367
[1] 4.00000000 0.01394367
[1] 4.10000000 0.01394367
[1] 4.20000000 0.01394367
[1] 4.30000000 0.01394367
[1] 4.40000000 0.01394371
[1] 4.50000000 0.01394371
```

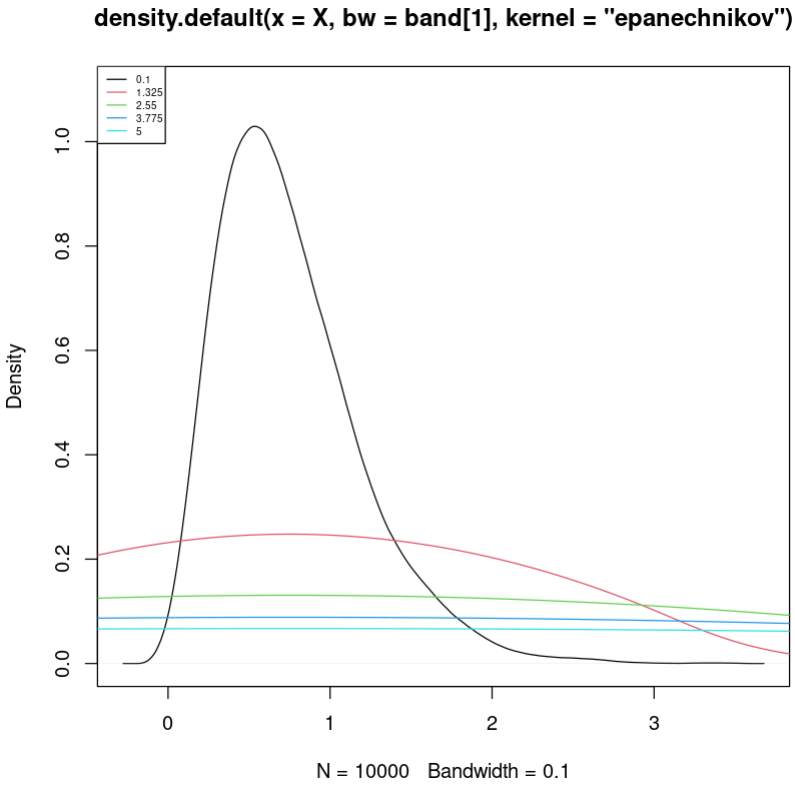
```
Error in value[[3L]](cond): Normalization error: The function will not i
ntegrate.Two common causes are: 1.) The kernel is non-smooth, try a smoo
th kernel if possible. 2.) The supplied support is incorrect.
Traceback:

1. kdensity(X, kernel = k1, bw = bandwidths[k])
2. tryCatch(stats::integrate(integrand, lower = support[1], upper = supp
ort[2]),
 .     error = function(e) {
 .         stop(paste0("Normalization error: The function will not integ
rate.",
 .             "Two common causes are: 1.) The kernel is non-smooth, ",
 .             "try a smooth kernel if possible. 2.) The supplied ",
 .             "support is incorrect."))
 .     })
3. tryCatchList(expr, classes, parentenv, handlers)
4. tryCatchOne(expr, names, parentenv, handlers[[1L]])
5. value[[3L]](cond)
6. stop(paste0("Normalization error: The function will not integrate.",
 .     "Two common causes are: 1.) The kernel is non-smooth, ",
 .     "try a smooth kernel if possible. 2.) The supplied ", "support is
incorrect."))
```

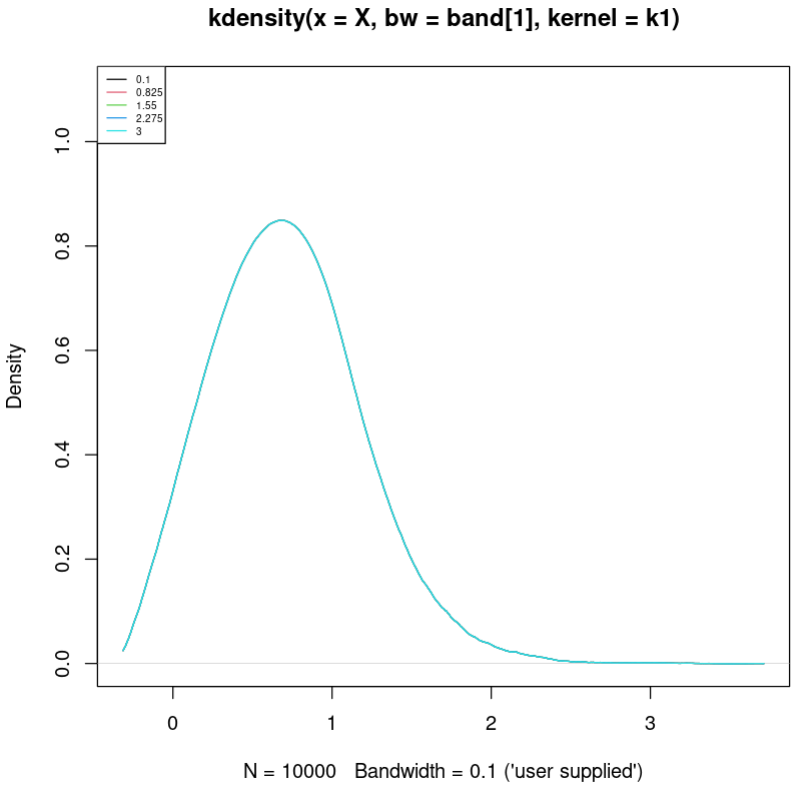
In [81]:
```
print('best_mise')
print(best_mise)
```

```
[1] "best_mise"
[1] 0.02086864
```

In [90]:
```
band=seq(from=0.1,to=5, length=5)
plot(density(X, kernel='epanechnikov', bw=band[1]),ylim=c(0, 1.1))
for (k in 2:length(band)){
  lines(density(X, kernel='epanechnikov', bw=band[k]),col=k)
}
legend("topleft",legend=band,col=1:length(band),cex=0.5,lty=1)
```



In [98]:
```
band=seq(from=0.1,to=3, length=5)
plot(kdensity(X, kernel=k1, bw=band[1]),ylim=c(0, 1.1))
for (k in 2:length(band)){
  lines(kdensity(X, kernel=k1, bw=band[k]),col=k)
}
legend("topleft",legend=band,col=1:length(band),cex=0.5,lty=1)
```



Почему-то мое самописное ядро работает направильно((

Задача 3

In [105…
```
install.packages("mixtools")
library(mixtools)
```

```
Retrieving 'https://packagemanager.rstudio.com/all/__linux__/focal/lates
t/src/contrib/mixtools_2.0.0.tar.gz' ...
        OK [downloaded 1.3 Mb in 0.3 secs]
Retrieving 'https://packagemanager.rstudio.com/all/__linux__/focal/lates
t/src/contrib/kernlab_0.9-32.tar.gz' ...
        OK [downloaded 2.1 Mb in 0.2 secs]
Retrieving 'https://packagemanager.rstudio.com/all/__linux__/focal/lates
t/src/contrib/segmented_1.6-2.tar.gz' ...
        OK [downloaded 741.6 Kb in 0.3 secs]
Installing kernlab [0.9-32] ...
        OK [installed binary]
Moving kernlab [0.9-32] into the cache ...
        OK [moved to cache in 0.28 milliseconds]
Installing segmented [1.6-2] ...
        OK [installed binary]
Moving segmented [1.6-2] into the cache ...
        OK [moved to cache in 0.38 milliseconds]
Installing mixtools [2.0.0] ...
        OK [installed binary]
Moving mixtools [2.0.0] into the cache ...
        OK [moved to cache in 0.27 milliseconds]
```

```
mixtools package, version 2.0.0, Released 2022-12-04
This package is based upon work supported by the National Science Founda
tion under Grant No. SES-0518772 and the Chan Zuckerberg Initiative: Ess
ential Open Source Software for Science (Grant No. 2020-255193).
```

In [100…
```
N=1000
eta=sample(1:6,size=N,prob=c(0.5,rep(0.1,5)),replace=TRUE)
m=c(0,seq(from=-1,to=1,by=0.5))
s=c(1,rep(0.1,5))
X=rep(NA,N)
for (k in 1:N){
  X[k]=rnorm(1,mean=m[eta[k]],sd=s[eta[k]])
}
```

In [151…
```
n_components_values = seq(from=2, to=10)
loglik_best = -10000
best_k = 0
for (k in 1:length(n_components_values)) {
        d=normalmixEM(X, k = n_components_values[k])
        if (d$loglik > loglik_best) {
                loglik_best = d$loglik
                best_comp =n_components_values[k]
        }
}
```

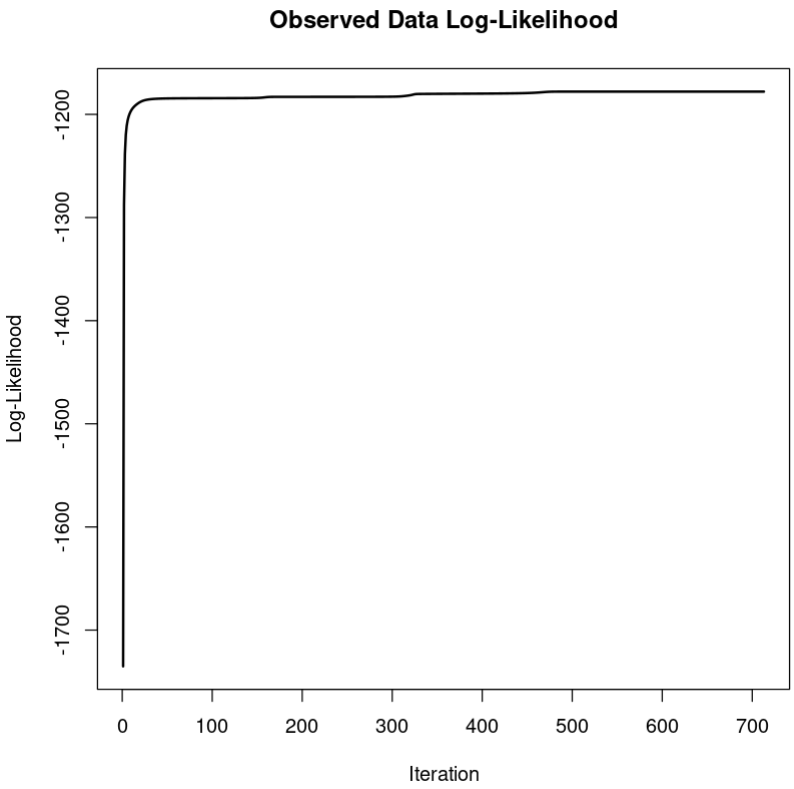
```
number of iterations= 33
number of iterations= 176
number of iterations= 792
number of iterations= 473
number of iterations= 229
WARNING! NOT CONVERGENT!
number of iterations= 1000
WARNING! NOT CONVERGENT!
number of iterations= 1000
WARNING! NOT CONVERGENT!
number of iterations= 1000
WARNING! NOT CONVERGENT!
number of iterations= 1000
```

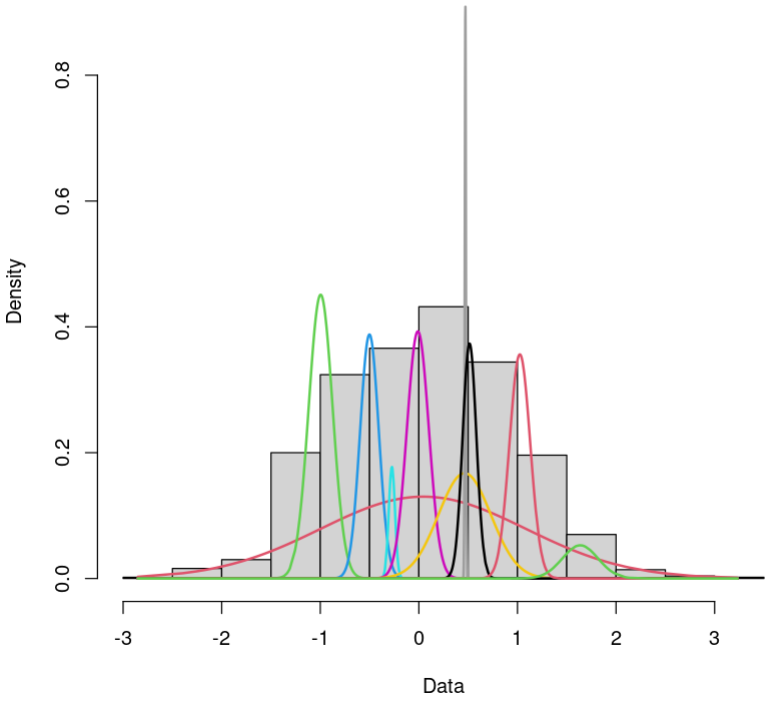
In [153…
```
print(c('best_component', 'best_loglik'))
print(c(best_comp, loglik_best))
```

```
[1] "best_component" "best_loglik"
[1]    10.000 -1179.351
```

In [154…
```
plot(normalmixEM(X, k = 10), density=T)
```

```
number of iterations= 712
```





Видно, что алгоритм переобучился